Genome sequences of SARS-CoV-2 as Data sets from CGR and FCGR for ML/DL Algorithms

Abstract

Genome sequencing has played a vital role in understanding the progress and genomic diversity of SARS-CoV-2 during this pandemic. The prediction is keen with Machine Learning (ML) and Deep Learning (DL) algorithms to find significant results in the field of genetics. In this study we analyse the genomic sequences of Covid-19 using the Chaos Game Representation (CGR) and Frequency Chaos Game Representation (FCGR) to prepare an efficient dataset for feeding input as datasets to ML/DL. As this research require a dataset to evaluate the frequency of nucleotides for a resourceful prediction result. Here, the dataset is composed of 100 set of nucleotide records collected from NCBI SARS-CoV-2 resources. By these findings it used as a reference for the machine learning or deep learning to forecast the influence of diseases 99% precisely and for understand the variations between past and present investigations.

Keywords— **DNA, nucleotide, Deep Learning, CGR**

I. INTRODUCTION

The identification in prototype of genome sequences is most essential ultimately for their recognized behaviours for analysis. In this work, identification of genome sequences and the k-mer has identified using a superior graphical method coined as Chaos Game Representation (CGR) and Frequency Chaos Game Representation (FCGR)[1-4]. Also, the k-mers are properly evacuated from the order as it has been represented in CGR, encountered tri-mer counts, fed into deep learning algorithms. It is necessary to recognize the k-mers they exhibit and to be able to deduce them in a biologically significant way and utilized the extracted k-mer patterns by calculating the mono, dinucleotide and trinucleotide frequencies [2].

This kind of processed data of genome sequence is essential to understand the utility of wide arrangements of genes. It is important to merge all sequence information in an obvious and efficient database to provide in a well-organized manner of sequence [5]. This processed genomic sequence data fed in to database will be provided as training and testing data for deep learning predictions. As the study of Covid-19 increases now-a-days, the predictions with Machine Learning and Deep Learning algorithms, will be utmost helpful in research to find significant results in the field of genetics. The sequences are downloaded from NCBI SARS-CoV-2 resources based on the parameters in datasets as we preferred. This dataset provides the numerical representation of the nucleotide of the sequences. All the sequences which are in FASTA format which analysed by Chaos Game Representation (CGR). Table II provides the complete specification about the genomic data acquired for this work.

II. SPECIFICATION TABLE

Subject	Genetics and Molecular Biology
Specific subject area	Bioinformatics
Type of data	Numerical (Nucleotide count)
How data were acquired	Official Genomic data Resources
Data format	1.Raw data are in text file (.fasta) 2.Analysed data in workbook (.csv)
Parameters for data collection	Amino acid combinations collected from genetic DNA sequence with CGR and FCGR
Description of data collection	The data were download from NCBI repository (SARS-CoV-2)
Data source location	Severe Acute Respiratory Syndrome Corona virus 2 isolate SARS-CoV-2/ human/IND/IGIB107037/2020
Data accessibility	Repository name: NCBI URL to data: https://www.ncbi.nlm.nih.gov/sars-cov-2/

Value of the data:

- The dataset provides a numerical data for SARS-CoV-2 nucleotide records which possible to find the variations between diseases.
- Thus, it used to detect the genomic diseases easily and efficiently.
- It is used as a reference to predict the diseases using the machine learning or deep learning algorithms.
- Researchers in the field of bioinformatics can benefit from these data because it is possible to use data stream.
- The Data scientists may utilize the results materialized from the quickly developing field of comparative genomics

III. DATA DESCRIPTION

This dataset shows the count of nucleotide using CGR [6][7]. The dataset contains about one hundred set of nucleotides records downloaded from (National Centre for Biotechnology Information) NCBI (SARS-CoV-2 Genbank) by sorting the parameters as we preferred. For instances, location, length, etc., The sequences are downloaded in the form of Fasta file (.fasta) in the root directory. In each file the information about the sequence were present in the top of the file. The csv format is used to store the counts of analysed nucleotide sequence for the sake of convenience.

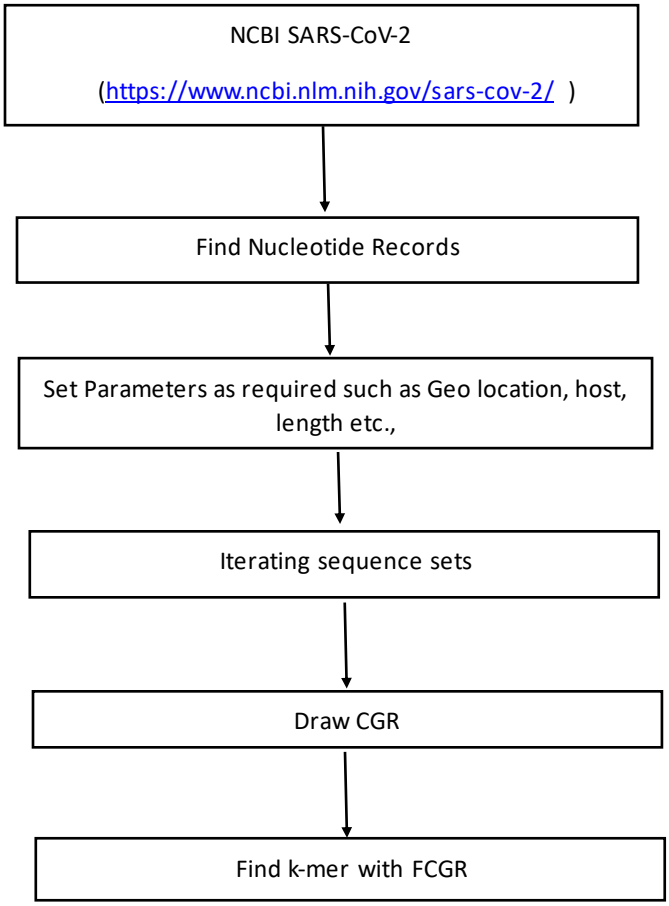

Fig 1: Flow for dataset making

IV. EXPERIMENTAL DESIGN

The Raw sequences were download from NCBI SARS-CoV-2 illustrated in Figure 1. For each file the numerical representation of nucleotide counts which is the dataset further given to DL or ML [8]. We utilize the proposed algorithm for CGR which gives the graphical structure of all sequences and the provides corresponding k-mer counts of the nucleotide with FCGR. While reading the sequence line by line it neglects the characters other than A, G, C, T and form a new long string of sequences [9][10]. A dictionary was created where the key as sequence and value as number of nucleotides in the dataset. The k-mer logic is used to detect the count of the nucleotide by calculating the mono-mer, di-mer, tri-mer nucleotide counts

where the tri-mer was given as the input for DL or ML. The mathematical expression for k-mer,

k-mer = N-k+1 - 1

where, N is the total number of sequence and k is size.

V. MATERIALS AND METHODS

The method CGR is used to create a geometrical form of genomic sequences. The procedure creates a square with Z_i coordinates in the form of A(0,0), T(10,0), C(0,1), G(1,2) for the 4 nucleotide in the genomic sequence. Draw a dot in the centre of the square at N_0= (0.5, 0.5) to act as the primary drawing mark [11][12]. Then draws a mark between the primary mark and the vertex corresponding to the first character from the input sequences (Z_i). The new mark (N_n+1) acts as the new start mark, and the procedure continues from there. A new mark is plotted between the mark in the beginning and the vertex corresponding to the next input character. The geometrical structure is generated by repeating this procedure until the last character of the genomic sequences is read and processed. The formula,

$$N_{n+1} = ½ (N_n + Z_i)$$

Where N_{n+1} is coordinate of new dot,
N_n is the coordinate of the present mark,
Z_i is the coordinate of the vertex corresponding to the nucleotide from the sequences.

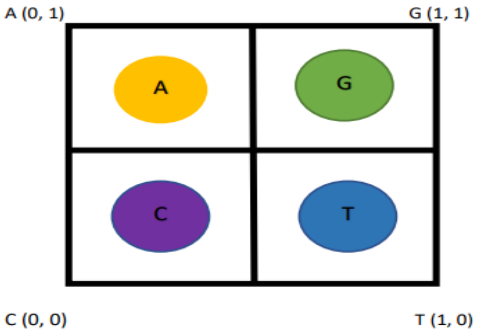

Fig 2: Areas of CGR

In the progress of FCGR, each nucleotide of the input DNA is inserted into the relevant nucleotide area's sub square. Each mark inside the region represents a substring of the DNA string that ends with that specific letter, and each square represents the area next to each vertex. The frequency of nucleotides equivalent in that area of the input string is represented by the number of marks in each area, which is known as the frequency chaos game representation [7][13].

The number of areas of CGR might be represented by a $2^n \times 2^n$ grid, where n denotes a group of n consecutive nucleotides (k-mers) inside the DNA string. Each square is split into 4^n sub-squares, with each sub-side square's being 2^{-n} in size. If n=2, for example, CGR will be split into 16 sub-squares.

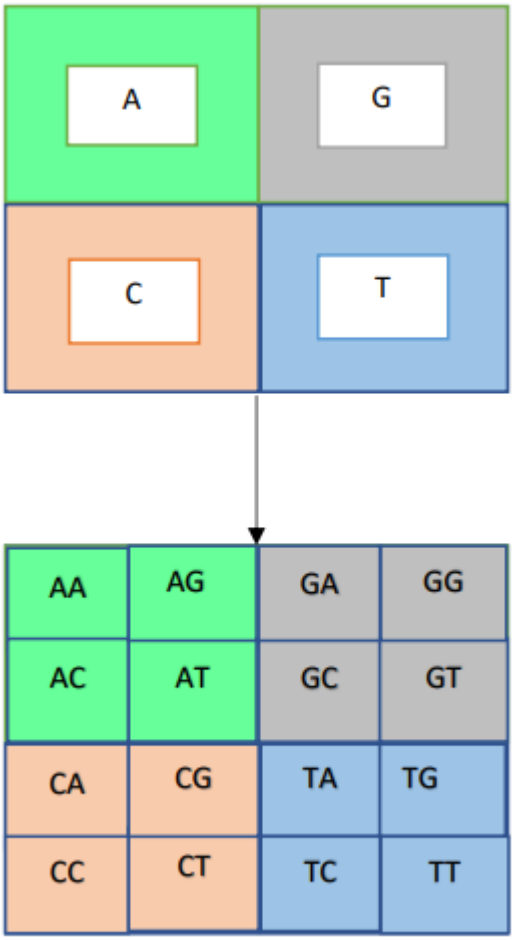

Fig3: CGR of IGIB107037/2020

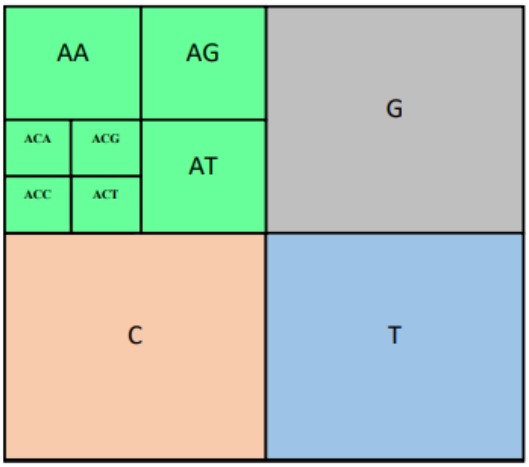

Fig 4: CGR for k-mer

MARKOV CHAIN MODEL

We used Markov chain model to find a random probability distribution describing the acquired genomic sequence of possible events in which the probability of each event depends on the previous event [2][10].

First-order Markov Chain model:

The first-order Markov chain model has brought out the di-nucleotides frequencies (n_{pq}) and the probabilities (P_{pq}) of the sequences which given in the table 2.

Second-order Markov chain model:

The second-order Markov chain model has brought out the Tri-nucleotides frequencies (n_{pqr}) and the probabilities (P_{pqr}) of the sequences which are given in the table 3.

Table II: mono-mer counts:

A	8890
G	9576
C	5461
T	5852

Table III: di-mer counts:

AA	2167
AT	1808
TA	1799
TT	2449
AC	1574
AG	1300
TC	1081
TG	2037

Table IV. di-mer probability:

AA	0.073
AT	0.061
TA	0.061
TT	0.083
AC	0.053
AG	0.044
TC	0.036
TG	*0.069*

Table V: tri-mer counts:

AAT	574
ATA	365
TAC	470
TTC	392
ACC	297
AGC	214

TCC	169
TGC	408

Table VI: tri-mer probability:

AAT	0.019
ATA	0.012
TAC	0.015
TTC	0.013
ACC	0.010
AGC	0.007
TCC	0.005
TGC	0.013

CGR GRAPH:

We obtained a unit square which produces a CGR of the acquired genomic data. Set starting point as center of a square (0.5,0.5). The first nucleotide is drawn half way between the starting point and the end point and the vertex corresponding to this nucleotide. Following then, for each nucleotide after that, a point is plotted as the halfway point between the previously plotted points and the vertex corresponding to the nucleotide. Figure 4 gives the CGR for the sequence "ACGT" from acquired genome [6][9].

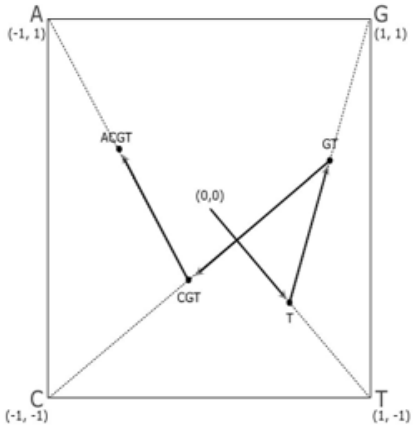

Fig 4: CGR graph for sequences "ACGT"

Figure 5 presented for the flow for multiple set of acquired genomic sequences conversion to dataset

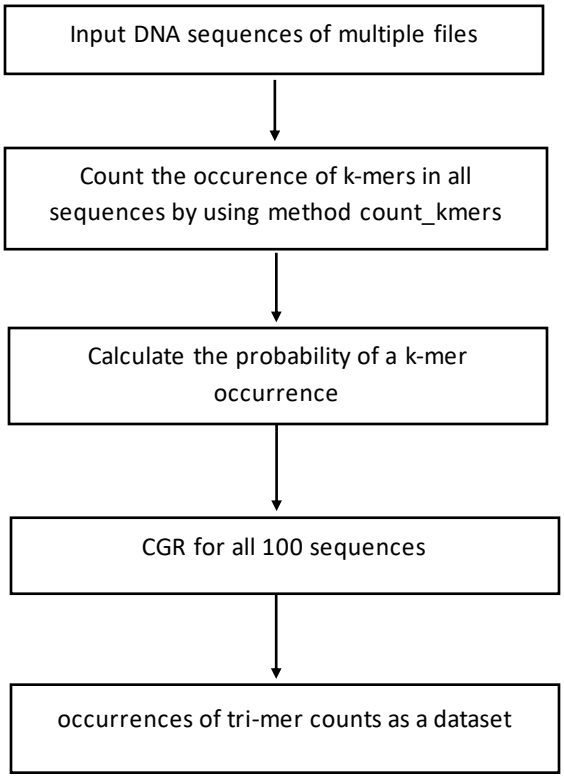

Fig 5: multiple file conversion flow

ALGORITHM:

After successful retrieval of Genomic sequences from NCBI, the following algorithm will be evaluated and datasets are produced.

Listing. 1

1. *Grouping into destination directory as S*
2. *Set n as file name*
3. *For n in S*
4. *Apply CGR code*
5. *Find k-mers (N-k+1)*
    a. *N – length of sequence*
    b. *k – occurrence*
6. *Apply FCGR*
7. *Separate tri-mer counts among total k-mers*
8. *Extract the results*

The proposed algorithm was implemented and simulated by a python programming code snippet. It is working based on the k-mer synthesis from FCGR.

JACCARD SIMILARITY:

The Jaccard similarity coefficient measures the similarity between two sets. It is defined as the size of the intersection divided by the size of the union of the sets. To calculate

the Jaccard similarity coefficient between the first sequence and a set of other sequences until n, we can follow these steps:

1. Convert the first sequence and each of the other sequences to sets of unique elements. Let the first sequence be set A, and the other sequences be sets B1, B2, ..., Bn.
2. Calculate the intersection between sets A and Bi for each i from 1 to n. Let the size of the intersection between A and Bi be denoted as $|A \cap Bi|$.
3. Calculate the union between sets A and Bi for each i from 1 to n. Let the size of the union between A and Bi be denoted as $|A \cup Bi|$.
4. Calculate the Jaccard similarity coefficient between A and Bi for each i from 1 to n using the formula:

$$\mathbf{J(A, Bi) = |A \cap Bi| / |A \cup Bi|}$$

The equation for Jaccard similarity coefficient between the first sequence and a set of other sequences until n can be written as:

$J(A, B1, B2, ..., Bn) = [\Sigma (|A \cap Bi| / |A \cup Bi|)] / n$

where Σ is the summation symbol, and n is the total number of other sequences being compared to the first sequence.

for each Seq2 in SeqSet:

Set2 = set(Seq2) # convert Seq2 into a set of unique elements

Int2 = set(Seq1) & Set2 # calculate the intersection of Seq1 and Set2

Uni2 = set(Seq1) | Set2 # calculate the union of Seq1 and Set2

Jaccard(Seq1, Seq2) = len(Int2) / len(Uni2) # calculate the Jaccard similarity coefficient.

NORMALIZED K MER COUNT:

Algorithm:

1. For each reference sequence Ri:

   a. Count the number of k-mers in Ri using a hash table or a similar data structure.

   b. Compute the total number of k-mers in Ri, denoted as Ni.

2. For each query sequence Qj:

   a. Count the number of k-mers in Qj using the same hash table or data structure used in step 1.a.

   b. Compute the total number of k-mers in Qj, denoted as Nj.

3. For each pair (Ri, Qj):

a. Count the number of k-mers shared by Ri and Qj using the same hash table or data structure used in step 1a and 2a.

b. Normalize the k-mer count by dividing it by the average k-mer count in Ri and Qj, which is (Ni + Nj) / 2.

c. Further normalize the k-mer count by dividing it by the normalization factor N.

d. Store the normalized k-mer count in M(i,j).

4. Return the matrix M.

```
FUNCTION count_kmers(seq, k):
    kmer_counts = empty dictionary
    FOR i FROM 0 TO len(seq) - k:
        kmer = seq[i:i+k]
        IF kmer is in kmer_counts:
            increment value of kmer in kmer_counts
        ELSE:
            add kmer to kmer_counts with value 1
    RETURN kmer_counts

seq1_counts = count_kmers(seq1, k)
normalized_counts = empty list
FOR seq in seqs:
    current_counts = count_kmers(seq, k)
    total_kmers = 0
    shared_kmers = 0
    FOR kmer, count in seq1_counts.items():
        total_kmers += count
        IF kmer is in current_counts:
            shared_kmers += min(count, current_counts[kmer])
    FOR count in current_counts.values():
        total_kmers += count
    normalized_count = shared_kmers / total_kmers
```

ADD normalized_count to normalized_counts

RETURN normalized_counts

WEIGHTED K-MER COUNT:

Here's an algorithm for computing the weighted k-mer count for comparing the first sequence with a set of other sequences, where n is the number of sequences in the set:

1. Set k equal to the desired length of the k-mers.
2. Create an empty dictionary called "counts".
3. For each sequence in the set (excluding the first sequence): a. Create an empty dictionary called "kmer_counts". b. Iterate over each k-mer of length k in the sequence: i. If the k-mer is not already in kmer_counts, add it as a key with a value of 1. ii. If the k-mer is already in kmer_counts, increment its value by 1. c. Compute the total count of k-mers in the sequence by summing the values in kmer_counts. d. For each k-mer in kmer_counts: i. If the k-mer is not already in counts, add it as a key with a value of the count of the k-mer in the sequence. ii. If the k-mer is already in counts, add the count of the k-mer in the sequence to its existing value.
4. Create an empty list called "weighted_counts".
5. Create an empty dictionary called "first_kmer_counts".
6. Iterate over each k-mer of length k in the first sequence: a. If the k-mer is not already in first_kmer_counts, add it as a key with a value of 1. b. If the k-mer is already in first_kmer_counts, increment its value by 1.
7. Compute the total count of k-mers in the first sequence by summing the values in first_kmer_counts.
8. For each k-mer in first_kmer_counts: a. If the k-mer is in counts, compute its weighted count by dividing its count in the first sequence by the total count in the first sequence, and then multiplying by the total count in the other sequences. b. If the k-mer is not in counts, set its weighted count to 0. c. Add the k-mer and its weighted count to weighted_counts.
9. Sort weighted_counts in descending order based on the weighted count.
10. Return weighted_counts.

This algorithm computes the weighted k-mer count for the first sequence compared to a set of other sequences. The weighted count takes into account the frequency of each k-mer in the first sequence and the frequency of each k-mer in the set of other sequences. The result is a list of k-mers sorted by their weighted count, where the k-mers that are more frequent in the first sequence relative to the other sequences are at the top of the list.

```
Function weighted_kmer_count(sequence, k, weights):
    kmer_counts = {}
    sequence_length = length(sequence)
    for i in range(sequence_length - k + 1):
        kmer = sequence[i:i+k]
        if kmer not in kmer_counts:
            kmer_counts[kmer] = 0
        kmer_counts[kmer] += weights[i]
    return kmer_counts

Function compare_sequences(sequence1, sequences, k, weights, n):
    sequence1_counts = weighted_kmer_count(sequence1, k, weights)
    results = []
    for i in range(n):
        sequence2_counts = weighted_kmer_count(sequences[i], k, weights)
        distance = 0
        for kmer, count in sequence1_counts.items():
            if kmer in sequence2_counts:
                distance += abs(count - sequence2_counts[kmer])
            else:
                distance += count
        for kmer, count in sequence2_counts.items():
            if kmer not in sequence1_counts:
                distance += count
        results.append(distance)
    return results
```

VI. RESULTS AND DISCUSSION

The dataset provided as a numerical data for SARS-CoV-2 nucleotide records which could feasible to identifying the variations between diseases. In this observation the datasets used to detect the genomic diseases by the prediction of ML/DL efficiently. Also used as a reference to predict the diseases using the various machine learning or deep learning algorithms.

Raw sequence data were processed with respective k-mers counts and progressively added to the repository according to the given algorithm (Listing 1). This dataset consists of (64 x n) 64 rows and n columns as the preferred inputs from NCBI data. Table VI illustrates the datasets extracted from genome sequences by CGR and FCGR. Above hundreds of data is executed to produce the data set with the help of above said algorithm for the testing purposes [8][14][15-17].

Table VI: complete dataset

Trimer	AAA	TTT	…				GGG
NC_102	543	674	…				975
NB_321	432	653					235
NG_542	865	345	…				754

We could extract any type of genomic sequences by CGR and FCGR methods and store the data as the above said format (.csv). It is not limited to Covid-19 genomes. The presented data might be used as references for any kind of genomic data [5][6][17][18].

VII. CONCLUSION

We analyzed the genomic sequences of Covid-19 using the Chaos Game Representation (CGR) and Frequency Chaos Game Representation (FCGR) and prepared an efficient dataset for feeding input as datasets to ML/DL. These fruitful predictions of diverse Machine Learning (ML) and Deep Learning (DL) algorithms are found to be given significant results from the tests. As this research required a dataset to evaluate the frequency of nucleotides for an effective prediction result, the dataset is composed of 100 set of nucleotide records collected from NCBI SARS-CoV-2 resources. In this observation the datasets prepared from CGR and FCGR used as a training/testing data for the machine learning or deep learning to forecast the influence of diseases 99% precisely and for understand the variations between past and present investigations.